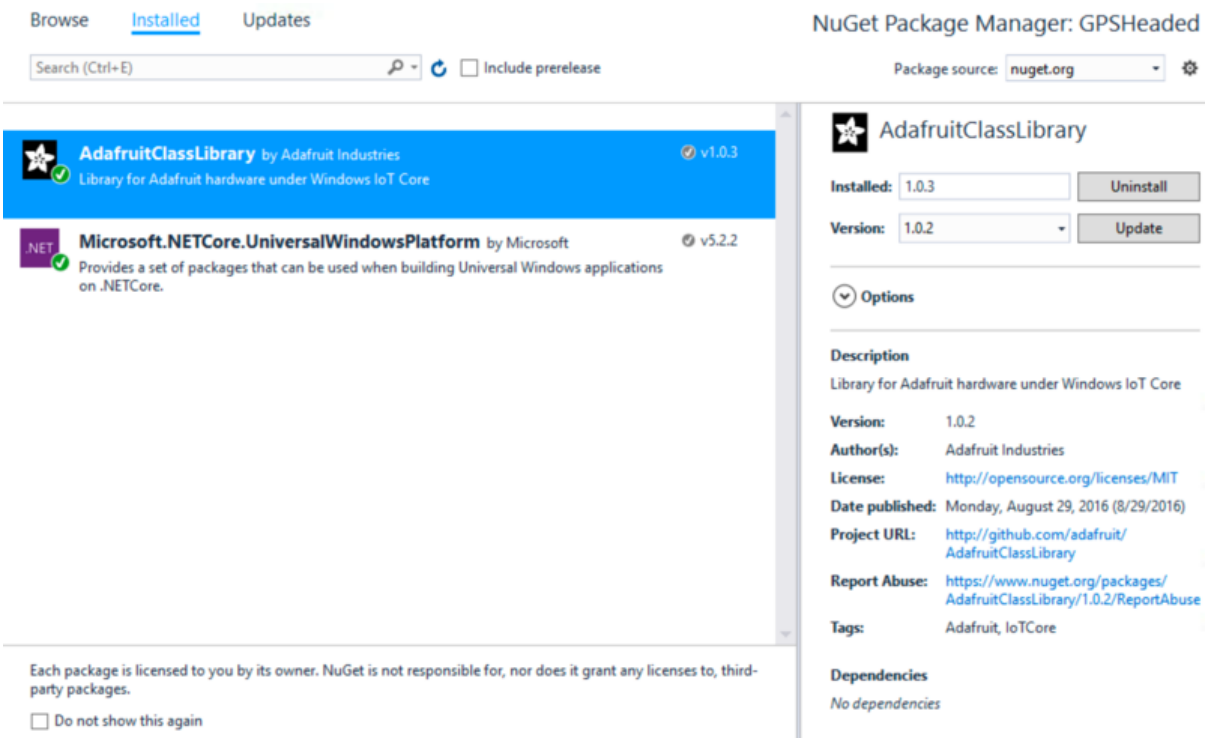
Adafruit Class Library for Windows IoT Core

Created by Rick Lesniak



https://learn.adafruit.com/adafruit-class-library-for-windows-iot-core

Last updated on 2023-08-29 03:13:57 PM EDT

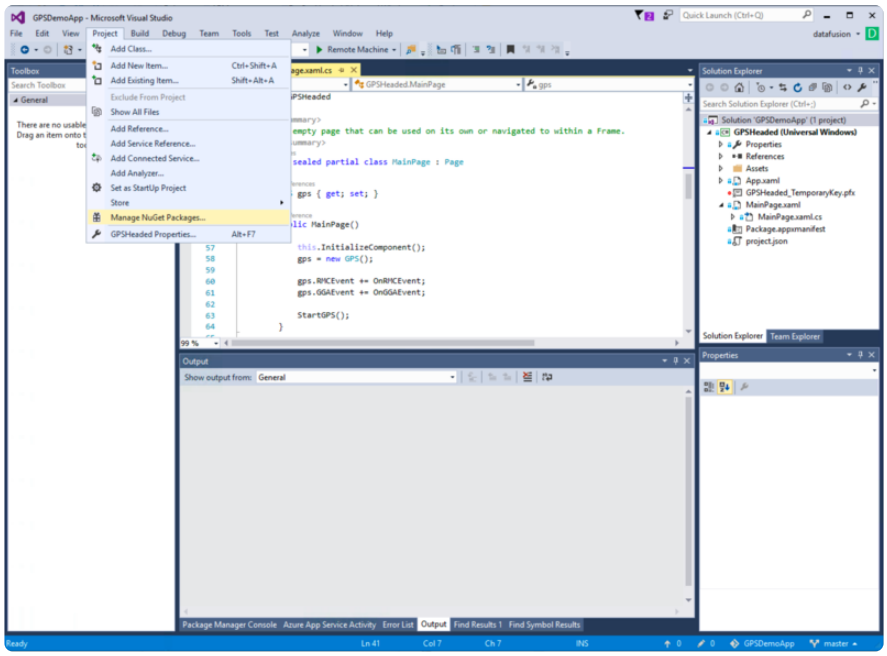
Table of Contents

Overview

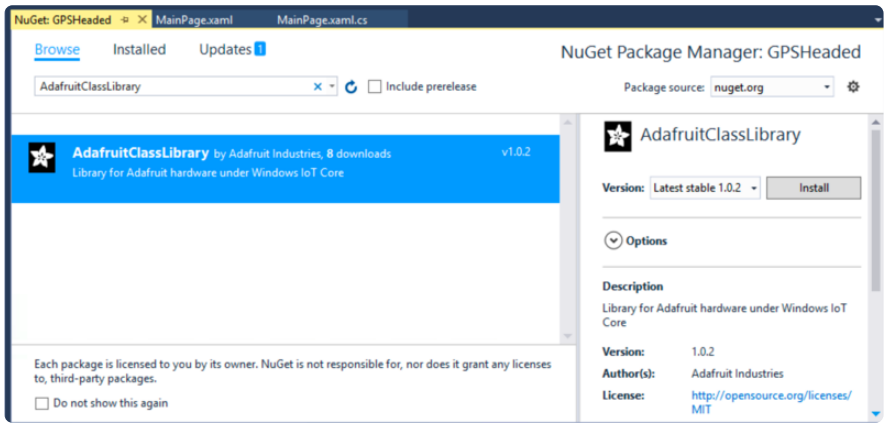
The Adafruit Class Library is a special library package containing Windows IoT Core driver software for a variety of Adafruit products. To use the library, you must add a reference to it in your project.

To add the reference to the Adafruit Class Library, you'll need to use the NuGet Package Manager, which is a standard part of Visual Studio.

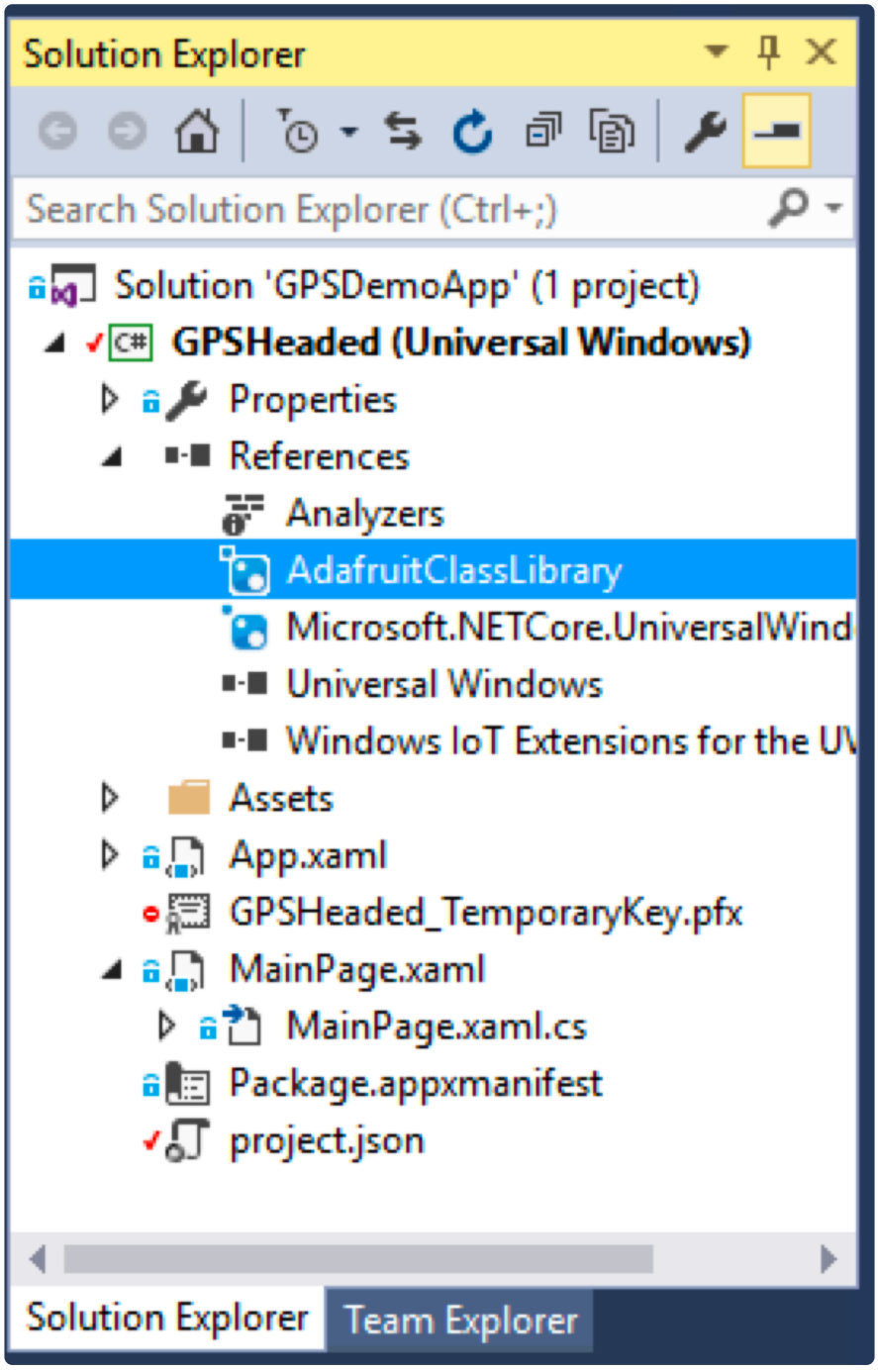
To get to the Package Manager, open the Project Menu and select "Manage NuGet Packages..."



In the Package Manager window, select "Browse", and enter "AdafruitClassLibrary" in the search box. Select the library in the list, and click the Install box on the right-hand side of the window



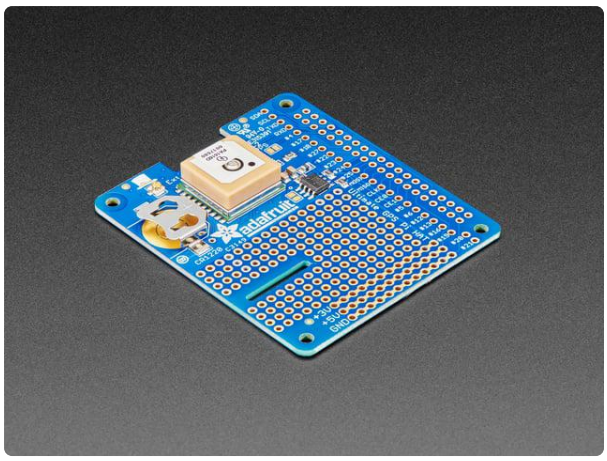
You should now see AdafruitClassLibrary under References in Solution Explorer. That's all there is to it!



Adafruit Class Library Sources

GPS Class

The GPS Class is designed to work with the Adafruit Ultimate GPS Hat. This page documents the public API of the GPS Class.



Adafruit Ultimate GPS HAT for Raspberry Pi A+/B+/Pi 2/3/Pi 4
It's 10PM, do you know where your Raspberry Pi is? If you had this GPS HAT, you would! This new HAT from Adafruit adds our celebrated Ultimate GPS on it, so you can add...
https://www.adafruit.com/product/2324

Constructor

The class has a single constructor, GPS(), which takes no arguments. Example:

```
GPS gps = new GPS();
```

Commands

The class supports several methods which write command strings to the GPS module.

SetSentencesReportingAsync

This is an asynchronous method with a return type of Task. SetSentencesReportingAsync issues the `PMTK314` command to the GPS module.

The method takes 6 arguments, each of which is the frequency of update for the corresponding NMEA sentence type. each argument takes an integer value of 0 to 5, where

0: for no sentence reporting
1: once every position fix
2: once for every two position fixes
3..5: once for every 3 to 5 position fixes

The arguments, in order, are

`GLLfreq`
`RMCfreq`
`VTGfreq`

`GGAfreq`
`GSAfreq`
`GSVfreq`

```
public async Task SetSentencesReportingAsync(int GLLfreq, int RMCfreq, int VTGfreq,
int GGAfreq, int GSAfreq, int GSVfreq)
```

SetUpdateFrequencyAsync

This is an asynchronous method with a return type of Task. SetUpdateFrequencyAsync issues the PMTK220 command to the GPS module.

The method takes a single argument of type double, which is the desired location reporting frequency in Hz. Values range from 0.1 to 10, where 0.1 is one location update every ten seconds, and 10 is 10 location updates per second.

```
public async Task SetUpdateFrequencyAsync(double freqHz)
```

SetBaudRateAsync

This is an asynchronous method with a return type of Task. SetBaudRateAsync issues the PMTK251 command to the GPS module.

The method takes a single argument of type unsigned integer, which is the desired baudrate setting for the GPS module. Note that this is the baudrate setting of the module itself, and does not affect the baud rate of the host computer.

```
public async Task SetBaudRateAsync(uint baudrate)
```

SendPMTKCommandAsync

This is an asynchronous method with a return type of Task. SendPMTKCommandAsync is a generic routine for sending an arbitrary PMTK command to the GPS module.

The method takes a single argument of type string, which is the text of the PMTK command. the command string must be a complete PMTK command, including the leading '$', the checksum, and the trailing characters.

See this link () for a PMTK checksum calculator.

```
public async Task SendPMTKCommandAsync(string pmtk)
```

Serial Control

Connected

A simple predicate property, taking no aruments, which returns true if the serial port on the host computer has been successfully opened. Note that this does not necessarily mean that communications with the GPS module have been established. Serial ports by nature are connectionless. This command assumes that the GPS is correctly attached to the host computer and is operational.

```
public bool Connected
```

ConnectToUARTAsync

This is an asynchronous method with a return type of Task. The method is used to open a serial port object on the hoat computer.

the method takes two optional arguments: a baudrate of type unsigned integer, and a UART ID of type string. Default values are 9600 baud and "UART0". "UART0" identifies the built-in Raspberry Pi serial port.

Call this method once to open a serial connection to the GPS module.

```
public async Task ConnectToUARTAsync(uint baudRate = 9600, string uartID = "UART0")
```

StartReading

This is a syncronous method that launches the GPS read task. After a successful call to ConnectToUART, call this method to begin reading GPS data.

```
public void StartReading()
```

StopReading

This is a syncronous method that cancels the GPS read task. Call this method to stop reading GPS data.

```
public void StopReading()
```

DisconnectFromUART

This is a synchronous method that stops reading the GPS and closes and disposes the serial port object. After calling this method, you must again call ConnectToUART and StartReading to resume receiving GPS data

```
public void DisconnectFromUART()
```

Data Classes

There is a data class associated with each type of NMEA sentence recognized by the GPS Class. Sentences are parsed and the results made available in objects of these types.

The GPS Class only supports the fiollowing NMEA sentence types: RMC, GGA, GLL, VTG, GSA, and GSV.

For details on the contents of these NMEA sentences, please see this page: http://aprs.gids.nl/nmea/ ()

GPSRMC

This is a data class defined within the GPS Class. Objects of this type are used to return parsed RMC sentence information to the host application. The following properties of the class are defined:

```
public DateTime TimeStamp { get; set; }
public bool Valid { get; set; }
public double? Latitude { get; set; }
public string LatHemisphere { get; set; }
public double? Longitude { get; set; }
public string LonHemisphere { get; set; }
public double? Speed { get; set; }
public double? Course { get; set; }
public DateTime DateStamp { get; set; }
public double? MagVariation { get; set; }
public string VarDirection { get; set; }
public double? LatDegrees { get; set; }
public double? LonDegrees { get; set; }
```

GPSGGA

This is a data class defined within the GPS Class. Objects of this type are used to return parsed GGA sentence information to the host application. The following properties of the class are defined:

```
public enum FixQuality { noFix = 0, gpsFix = 1, dgpsFix = 2 }

public DateTime TimeStamp { get; set; }
public double? Latitude { get; set; }
public string LatHemisphere { get; set; }
public double? Longitude { get; set; }
public string LonHemisphere { get; set; }
public FixQuality Quality { get; set; }
public int? Satellites { get; set; }
public double? Dilution { get; set; }
public double? Altitude { get; set; }
public string AltUnits { get; set; }
public double? Geoidal { get; set; }
public string GeoidalUnits { get; set; }
public double? DGPSAge { get; set; }
public int? DGPS_ID { get; set; }
public double? LatDegrees { get; set; }
public double? LonDegrees { get; set; }
```

GPSGLL

This is a data class defined within the GPS Class. Objects of this type are used to return parsed GLL sentence information to the host application. The following properties of the class are defined:

```
public DateTime TimeStamp { get; set; }
public bool Valid { get; set; }
public double? Latitude { get; set; }
public string LatHemisphere { get; set; }
public double? Longitude { get; set; }
public string LonHemisphere { get; set; }
public double? LatDegrees { get; set; }
public double? LonDegrees { get; set; }
```

GPSVTG

This is a data class defined within the GPS Class. Objects of this type are used to return parsed VTG sentence information to the host application. The following properties of the class are defined:

```
public double? TrackTrue { get; set; }
public string TT { get; set; }
public double? TrackMag { get; set; }
public string TM { get; set; }
public double? SpeedKnots { get; set; }
public string SKn { get; set; }
public double? SpeedKm { get; set; }
public string SKm { get; set; }
public string Mode { get; set; }
```

GPSGSA

This is a data class defined within the GPS Class. Objects of this type are used to return parsed GSA sentence information to the host application. The following properties of the class are defined:

```
public enum FixType { noFix = 1, fix2D = 2, fix3D = 3 }

public string Mode { get; set; }
public FixType Fix { get; set; }
public List&lt;int?&gt; SVIDs { get; set; }
public double? PDOP { get; set; }
public double? HDOP { get; set; }
public double? VDOP { get; set; }
```

GPSGSV

This is a data class defined within the GPS Class. Objects of this type are used to return parsed GSV sentence information to the host application. The following properties of the class are defined:

```
public int? MsgCount { get; set; }
public int? MsgNumber { get; set; }
public int? Satellites { get; set; }
public List&lt;SVRecord&gt; SVList { get; set; }
```

The SVList property contains a list of SVRecord objects The SVRecord class is defined within the GPSGSV class.

The list may be up to 4 elements in length. The properties of the SVRecord class are as follows:

```
public int? PRN { get; set; }
public int? Elevation { get; set; }
public int? Azimuth { get; set; }
public int? SNR { get; set; }
```

Event Handlers

When an NMEA sentence is successfully received and parsead, the GPS Class issues an event containing the data class associated with that sentence. The events are as follows:

```
public event RMCEventHandler RMCEvent;
public event GLLEventHandler GLLEvent;
public event VTGEventHandler VTGEvent;
public event GGAEventHandler GGAEvent;
```

```
public event GSAEventHandler GSAEvent;
public event GSVEventHandler GSVEvent;
```

To use these events, define an event handler in your code, and add that handler to the event list. Event handlers take two aruments, the sending object, and the sentence data class.

A sample event handler follows:

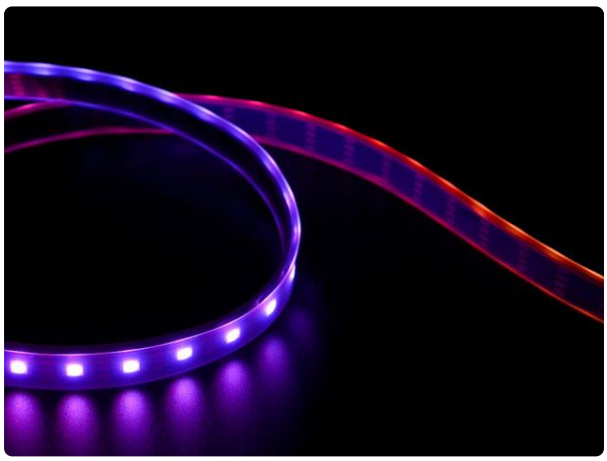
```
private void OnGGAEvent(object sender, GPS.GPSGGA GGA)
{
    if (GGA.Quality != GPS.GPSGGA.FixQuality.noFix)
    {
        AltitudeTextBox.Text = GGA.Altitude.ToString();
        SatellitesTextBox.Text = GGA.Satellites.ToString();
        AltUnitsTextBox.Text = GGA.AltUnits;
    }
    else
    {
        AltitudeTextBox.Text = "";
        SatellitesTextBox.Text = "";
        AltUnitsTextBox.Text = "";
    }
}
```

Associate your handler with the event by adding it to the event list:

```
gps.GGAEvent += OnGGAEvent;
```

DotStar Class

The DotStar Class is designed to work with Adafruit DotStar addressible digital RGB LEDs This page documents the public API of the DotStar Class.



Adafruit DotStar Digital LED Strip - Black 60 LED - Per Meter

Move over NeoPixels, there's a new LED strip in town! These fancy new DotStar LED strips are a great upgrade for people who have loved and used NeoPixel strips for a few years but...

https://www.adafruit.com/product/2239

Constructors

The DotStar class uses SPI to communicate wit the dotstars. The class has two constructors, one for Hardware SPI mode, and one for Software SPI mode. Hardware SPI uses the built-in SPI support of the Raspberry Pi, and so must operate from the SCLK and MOSI GPIO pins.

Software SPI mode can operate on any two GPIO pins, but is much slower than hardware SPI mode.

The hardware SPI constructore takes two arguments: the number of pixels and the color order of the pixels. Color order determines the sequence of RGB color values within the data streem to the Dot Stars. The color order argument is optional, and defaults to Blue,Red,Green.

```
public DotStar(uint numPixels, UInt32 colorOrder = DOTSTAR_BRG)
```

The software SPI constructor takes four arguments: The number of pixels, the GPIO number of the data pin, athe GPIO number of the clock pin, and the color order. Again, the color order argument is optional and defaults to Blue,Red,Green.

```
public DotStar(uint numPixels, int dataPin, int clockPin, UInt32 colorOrder =
DOTSTAR_BRG)
```

Methods

BeginAsync

The Begin method is an asynchronous method which initializes the SPI interface. It takes no arguments. Call Begin before attempting to write to the DotStars.

```
public async Task BeginAsync()
```

End

The End method shuts down the SPI interface. It takes no arguments.

```
public void End()
```

SetPixelColor

The SetPixelColor method assigns an RGB color to a specific pixel.

There are two overloads of SetPixelColor. The first takes an integer pixel number, and individual R, G, and B values as arguments, where R, G, and B are unsigned byte values in the range 0..255.

The second overload takes a pixel number and a 32-bit unsigned integer containing packed R, G, and B values.. Use the Color method to generate a packed value from individual R, G, B values.

Note that SetPixelColor does not update the displayed colors. It only set internal program storage. You must call the Show method to display the updated colors.

```
public void SetPixelColor(int pixel, byte r, byte g, byte b)
public void SetPixelColor(int pixel, UInt32 c)
```

GetPixelColor

The GetPixelColor method returns a 32-bit unsigned integer containing the packed R, G, B values for an indiviual pixel. It takes a single integer argument, which is the pixel number.

```
public UInt32 GetPixelColor(int pixel)
```

Clear

The Clear method turns off all pixels Ii.e., sets the color values to 0). It takes no arguments.

```
public void Clear()
```

Color

The color method converts individual R, G, Bvalues to a packed 32-bit unsigned integer. It takes integer R, G, B values as arguemnts. R,G, B values must be in the range 0..255.

```
public UInt32 Color(uint r, uint g, uint b)
```

UpdateLength

The UpdateLength method changes the number of pixels in the DotStar strip. It takes a single integer argument; the number of pixels in the strip.

As a side-effect, all previous pixel values are lost.

```
public void UpdateLength(uint numPixels)
```

Show

the Show method draws all stroed pixel values to the strip. The SetPixelColor methods only update internal program storage - they do not display the updated colors. You must call Show to display the updated colors.

Show takes no arguments.

```
public void Show()
```

CharLCDPlate Class

The CharLCDPlate Class is designed to work with the Adafruit Character LCD Plate. This page documents the public API of the GPS Class.

Note: the CharLCDPlate Class is dependent on the MCP23017 Class.

Adafruit RGB Positive 16x2 LCD+Keypad Kit for Raspberry Pi
This new Adafruit Pi Plate makes it easy to use an RGB 16x2 Character LCD. We really like the RGB Character LCDs we stock in the shop. (For RGB we have https://www.adafruit.com/product/1109

Adafruit RGB Negative 16x2 LCD+Keypad Kit for Raspberry Pi

This new Adafruit Pi Plate makes it easy to use an RGB 16x2 Character LCD. We really like the RGB Character LCDs we stock in the shop. (For RGB we have https://www.adafruit.com/product/1110

Constructor

The class has a single constructor, CharLCDPlate(), which takes no arguments.

The constructor creates an instance of the MCP23017 class.

```
public CharLCDPlate()
```

Methods

BeginAsync

This is an asynchronous method that initializes the MCP23017 and the Character LCD.

The method takes three integer arguments (the third argument is optional): cols, which specifies the number of columns on the LCD, lines, which specifies the number of lines on the LCD, and the optional argument dotsize, which specifies the dor matrix size of each character. Default is 5x8.

Constant values for the third argument are defined:

- public const byte LCD_5x10DOTS = 0x04;
- public const byte LCD_5x8DOTS = 0x00;

```
public async Task BeginAsync(int cols, int lines, int dotsize = LCD_5x8DOTS)
```

clear

This method takes no arguments and clears the LCD display

```
public void clear()
```

home

This method takes no arguments and moves the cursor to the 0,0 (home) position

```
public void home()
```

setCursor

This methof positions the cursor at the specified column and line. It takes two integer arguments, col and line.

```
public void setCursor(byte col, byte line)
```

createChar

The createChar method allows the host program to define a custom character. Up to 8 custom characters can be defined.

The method takes as arguments a location and an array of bytes. The location is the number of the custom character (0 through 7). The array is a set of bfitmapped bytes representing the dots of the character. For more information on defining custom characters, please see this page ().

```
public void createChar(byte location, byte[] charmap)
```

readButtons

This method reads the state of the 5 buttons on the plate. It retutns a byte containing a bitmap of the button states (1=pressed/0=unpressed).

The class contains bitmask definitions for the 5 buttons:

- public const byte BUTTON_UP = 0x08;
- public const byte BUTTON_DOWN = 0x04;

- public const byte BUTTON_LEFT = 0x10;
- public const byte BUTTON_RIGHT = 0x02;
- public const byte BUTTON_SELECT = 0x01;

```
public byte readButtons()
```

setBacklight

This method controls the color of the RGB backlight on LCDs with a color backlight. It takes a single integer argument, color, which is a bitmap of the three colors.

- Red : 0x04
- Green : 0x02
- Blue : 0x01

Backlights are either on or off. There is no shading.

```
public void setBacklight(int color)
```

print

There are three overloads of the print method. The first overload takes a string as an argument, the second overload takes an integer as an argument, and the third overload takes a double as an argument.

All three overloads write characters to the display beginning at the current cursor location.

```
public void print(string str)
public void print(int number)
public void print(double number)
```

Additional methods

The following methods are also available:

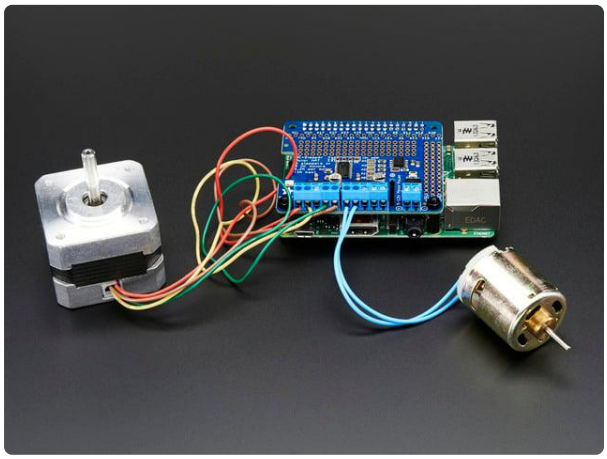
- noDisplay : turn display off
- display : turn display on
- noCursor : turn underline cursor off
- cursor : turn underline cursor on
- noBlink : turn cursor blink off

- blink : turn cursor blink on
- scrollDisplayLeft : scroll display left
- scrollDisplayRight : scroll display right
- leftToRight : sets text to flow from left to right
- rightToLeft : sets text to flow from right to left
- autoscroll : when printing, characters fill from right (right justify)
- noAutoscroll: when printing, characters fill from left (left justify)

```
public void noDisplay()
public void display()
public void noCursor()
public void cursor()
public void noBlink()
public void blink()
public void scrollDisplayLeft()
public void scrollDisplayRight()
public void leftToRight()
public void rightToLeft()
public void autoscroll()
public void noAutoscroll()
```

MotorHat Class

The MotorHat Class is designed to work with the Adafruit DC & Stepper Motor Hat. This page documents the public API of the MotorHat Class.



Adafruit DC & Stepper Motor HAT for Raspberry Pi - Mini Kit
Let your robotic dreams come true with the new DC+Stepper Motor HAT from Adafruit. This Raspberry Pi add-on is perfect for any motion project as it can drive up to 4 DC or 2 Stepper...
https://www.adafruit.com/product/2348

Constructors

The MotorHat Class has a single constructor. The constructor takes an optional argument of an I2C address for the PCA9685 chip the Hat is based on. The I2C addresses for the Hat is 0x60.

If no argument is provided the constructor configures the class to use the default I2C address of 0x60.

The MotorHat class references the PCA9685 class as superclass. The I2C address in the constructor is passed to the superclass.

The constructor allocates 4 objects of type DCMotor Class, and two objects of type StepperMotor Class. These classes are used to control the motors attached to the motor ports on the Hat. Please see the API descriptions below for the DCMotor and Stepper classes.

```
public MotorHat(int i2cAddr = 0x60) : base(i2cAddr)
```

Types

Enumerations

the PinState enum is used for setting PWM pins to a logic state of full off (LOW) or full on (HIGH)

```
public void SetPin(int pin, PinState state)
```

Methods

InitAsync

The InitAsync method takes a single argument which is the the PWM frequency for the PCA9685 chip. The method initializaes the PCA9685, sets the PWM frequency, and sets the PWM for all pins to 0.

```
public async Task InitAsync(uint freq)
```

SetPWM

Sets the PWM value for a specified pin. PWM values should be in the range 0 to 4096 inclusive.

```
public void SetPWM(int pin, ushort value)
```

SetPin

Sets the logic state for a specified pin. Logic states are either 0 (off) or 4096 (on). Uses the PinState enum. Values are LOW or HIGH.

```
public enum PinState { LOW, HIGH };
```

GetMotor

Returns an instance of the DCMotor class for the specified motor port. Motor port numbers are in the range 1 to 4, inclusive.

```
public DCMotor GetMotor(int index)
```

GetStepper

Returns an instance of the Stepper class for the specified stepper port. Stepper port numbers are in the range 1 to 2, inclusive.

Also takes the number of steps per revolution of the stepper attached to the port. The steps per revolution argument is ignored if GetStepper has already been called for the specified stepper port.

```
public Stepper GetStepper(ushort steps, int index)
```

DCMotor Class

The DCMotor class controls a single DC motor on one of the Motor Hat's 4 DC motor ports.

Types

The class has a single public enumeration, Command, which specifies the motion to be used by the motor: FORWARD, BACKWARD, and RELEASE.

RELEASE stops the motor.

```
public enum Command { FORWARD, BACKWARD, BRAKE, RELEASE };
```

Methods

SetSpeed

SetSpeed specifies the speed for the motor. Values range from 0 to 255, with 255 being the fastest. The value maps into a PWM value for the PCA9685, and is not an RPM value.

```
public void SetSpeed(uint speed)
```

Run

Run specifies the rotation of the motor, using the enum Command. the motor can be set to run FORWARD or BACKWARD. RELEASE stops the motor. the enum value BRAKE is not used and will be ignored.

```
public void Run(Command cmd)
```

Stepper Class

The Stepper class controls a single stepper motor on one of the Motor Hat's 2 stepper motor ports.

Types

The class has a two public enumerations, Command and Style.

Command declares values which specifiy the direction of motion to be used by the motor: FORWARD and BACKWARD.

Style declares values which specify the type of steps to be used by the motor. SINGLE, DOUBLE, INTERLEAVE, and MICROSTEP.

```
public enum Command { FORWARD, BACKWARD };
public enum Style { SINGLE, DOUBLE, INTERLEAVE, MICROSTEP };
```

Methods

SetSpeed

SetSpeed specifies the speed for the motor. Values are in RPM. Maximum RPM depends on the motor and the number of steps per revolution.

```
public void SetSpeed(uint rpm)
```

Release

Release stops the motor.

```
public void Release()
```

step

The step method specifies the number of steps to be taken, the direction of the steps, and the style of stepping. See the Command and Style enumeration types.

```
public void step(ushort steps, Command direction, Style style)
```

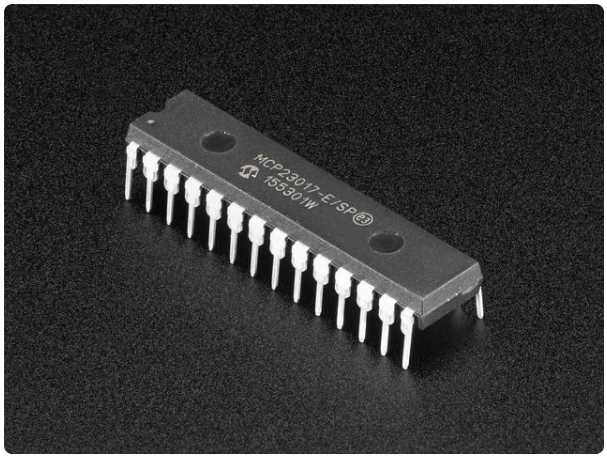
OneStep

The OneStep method advances the motor by one step in the specified direction and style of stepping. See the Command and Style enumeration types.

```
public int OneStep(Command direction, Style style)
```

MCP23017 Class

The MCP23017 Class provides a programming interface to the MCP23017 I2C port expander chip.



MCP23017 - i2c 16 input/output port expander

Add another 16 pins to your microcontroller using an MCP23017 port expander. The MCP23017 uses two i2c pins (these can be shared with other i2c devices), and in exchange gives you 16...

https://www.adafruit.com/product/732

Constructors

The MCP23017 Class has a single constructor. The constructor takes an optional argument of an I2C address. I2C addresses for the chip are in the range 0x20 to 0x27.

If no argument is provided the constructor configures the class to use the default I2C address of 0x20 for the MCP23017.

The MCP23017 class references the I2CBase class as superclass. The I2C address in the constructor is passed to the superclass

```
public MCP23017(int addr) public MCP23017(int addr = MCP23017_ADDRESS) :base(addr)
```

Methods

InitMCP23017Async

This is an asynchronous method whih initializes the I2C interface by calling the I2CBase Class InitI2CAsync method, and sets all MCP23017 GPIO pins to outputs.

```
public async Task InitMCP23017Async()
```

pinMode

The PinMode method sets the input/output direction of a single GPIO pin. It takes two arguments, the pin number, andthe direction.

Pin number is an integer in the range 0..15, where pins numbered from 0 to 7 are on Port A, and pins numbered from 8 to 15 are on Port B.

The direction argument is of type Direction, which is an enumeration defined within the class. Defined values are "INPUT" and "OUTPUT"

```
public enum Direction { INPUT = 0, OUTPUT = 1 };
public void pinMode(int p, Direction d)
```

pullup

This method sets the pullup state of a single GPIO pin to logical HIGH or LOW. It takes two arguments, a pin number and a logic value.

Pin number is an integer in the range 0..15, where pins numbered from 0 to 7 are on Port A, and pins numbered from 8 to 15 are on Port B.

The logic value argument is of type Level, which is an enumeration defined within the class. Defined values are "LOW" and "HIGH"

```
public enum Level { LOW = 0, HIGH = 1 };
public void pullUp(int pin, Level d)
```

digitalWrite

This method sets the state of a single GPIO pin to logical HIGH or LOW. It takes two arguments, a pin number and a logic value.

Pin number is an integer in the range 0..15, where pins numbered from 0 to 7 are on Port A, and pins numbered from 8 to 15 are on Port B.

The logic value argument is of type Level, which is an enumeration defined within the class. Defined values are "LOW" and "HIGH"

```
public enum Level { LOW = 0, HIGH = 1 };
public void digitalWrite(int pin, Level d)
```

digitalRead

This method gets the state of a single GPIO pin, and returns it as a logical HIGH or LOW. It takes a single arguments, a pin number.

Pin number is an integer in the range 0..15, where pins numbered from 0 to 7 are on Port A, and pins numbered from 8 to 15 are on Port B.

The return value is of type Level, which is an enumeration defined within the class. Defined values are "LOW" and "HIGH"

```
public enum Level { LOW = 0, HIGH = 1 };
public Level digitalRead(int pin)
```

writeGPIOAB

This writeGPIOAB method writes to all pins simultaneously. It takes a single argument, an unsigned 16-bit integer representing a bitmask of the desired logic state for each pin. The LSB corresponds to Port A, pin 0, and the MSB corresponds to Port B, pin 7.

```
public void writeGPIOAB(UInt16 ba)
```

readGPIOAB

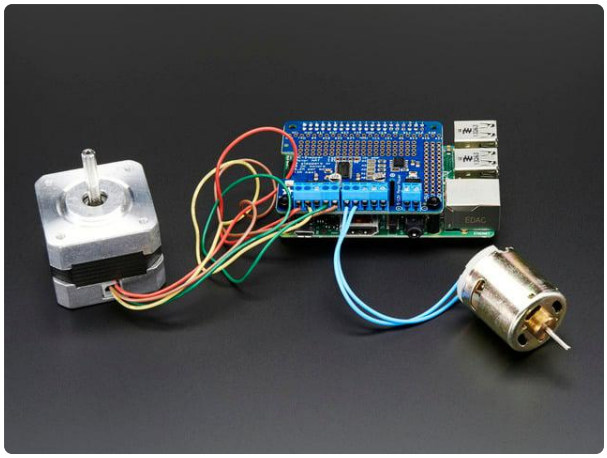
This readGPIOAB method reads all pins simultaneously. It returns a single value, an unsigned 16-bit integer representing a bitmask of the current logic state for each pin. The LSB corresponds to Port A, pin 0, and the MSB corresponds to Port B, pin 7.

```
public UInt16 readGPIOAB()
```

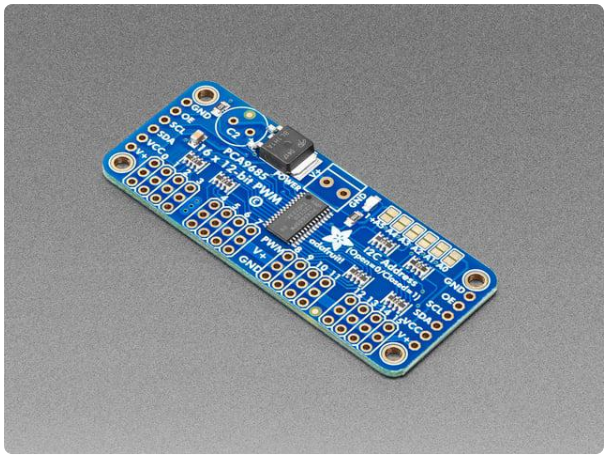
PCA9685 Class

The PCA9685 Class provides a programming interface to the PCA9685 I2C PWM/ Servo driver chip.

The PCA9685 is used in the Adafruit DC&Stepper Motor Hat and in the adafruit 16-channel 12-bit PWM/Servo Driver breakout.



Adafruit DC & Stepper Motor HAT for Raspberry Pi - Mini Kit
Let your robotic dreams come true with the new DC+Stepper Motor HAT from Adafruit. This Raspberry Pi add-on is perfect for any motion project as it can drive up to 4 DC or 2 Stepper...
https://www.adafruit.com/product/2348



Adafruit 16-Channel 12-bit PWM/Servo Driver - I2C interface
You want to make a cool robot, maybe a hexapod walker, or maybe just a piece of art with a lot of moving parts. Or maybe you want to drive a lot of LEDs with precise PWM output. Then...
https://www.adafruit.com/product/815

Constructors

The PCA9685 Class has a single constructor. The constructor takes an optional argument of an I2C address. I2C addresses for the chip are in the range 0x40 to 0x7F.

If no argument is provided the constructor configures the class to use the default I2C address of 0x40.

The PCA9685 class references the I2CBase class as superclass. The I2C address in the constructor is passed to the superclass

```
public PCA9685(int addr = PCA9685_ADDRESS) : base (addr)
```

Methods

InitPCA9685Async

This is an asynchronous method which initializes the I2C interface by calling the I2CBase Class InitI2CAsync method, and resets the chip

The I2CSpeed enumeration is inherited from I2CBase.

```
public async Task InitPCA9685Async(I2CSpeed i2cSpeed = I2CSpeed.I2C_100kHz)
```

Reset

Resets the PCA9685

```
public void Reset()
```

SetPWMFrequency

Sets the PWM frequency of the PCA9685. Takes a single argument of type double, in the range of 0.0 to 4095.0.

```
public void SetPWMFrequency(double freq)
```

SetPWM

Sets the PWM on a specified pin. Arguments are the pin number, the on time, and the off time.

```
public void SetPWM(int num, ushort on, ushort off)
```

SetAllPWM

Sets a PWM on all pins. Arguments are the on time and the off time.

```
public void SetAllPWM(ushort on, ushort off)
```

SetPin

Sets pin without having to deal with on/off tick placement and properly handles a zero value as completely off. Optional invert parameter supports inverting the pulse for sinking to ground. Parameters are pin number, PWM value, and a boolean invert.

PWM value should be from 0 to 4095 inclusive

```
public void SetPin(int num, ushort val, bool invert)
```

I2CBase Class

I2CBase serves as a common superclass for I2C device classes. Subclasses inherit the InitI2CAsync method and the I2CSpeed enumeration type.

InitI2CAsync must be called before any other I2C methods are called.

I2cSpeed

Sets the I2C clock speed. The speed can be set to either 100kHz (standard mode) or 400kHz (fast mode).

```
public enum I2CSpeed { I2C_100kHz, I2C_400kHz };
```

InitI2CAsync

The InitI2CAsync is an asynchronous method that creates and initializes a Windows Core IoT I2C device object. It takes a single optional argument of type I2CSpeed. Default is standard mode (100kHz).

```
public async Task InitI2CAsync(I2CSpeed i2cSpeed = I2CSpeed.I2C_100kHz)
```